# Formalizing Mechanical Analysis of Sweeping Nets IV

Parker Emmerson

October 2024

**Abstract**

In previous works, the sweeping net method has been applied to approximate singularities on two-dimensional manifolds and extended to higher-dimensional manifolds with isolated singularities. In this paper, we further formalize the mechanics of sweeping nets, proving new theorems on their properties and applications. We provide detailed proofs for each theorem, enhancing the mathematical foundation of the sweeping net method and broadening its applicability.

## Contents

**1 Introduction** **1**

**2 Conformality and Higher-Order Approximations** **1**
2.1 Theorem 31: Conformality of the Sweeping Nets . . . . . 1
2.2 Theorem 32: Higher-Order Approximations Improve Accuracy . . . . . 5

**3 Adaptive Mesh Refinement and Boundary Singularities** **5**
3.1 Theorem 33: Adaptive Mesh Refinement Improves Approximation Efficiency . . . . . 5
3.2 Theorem 34: Extension to Manifolds with Boundary Singularities . . . . . 8

**4 Computational Implementation and Applications** **8**
4.1 Theorem 35: Efficient Computational Implementation via Parallel Processing . . . . . 8

**5 Conclusion** **11**

# 1 Introduction

Sweeping nets have been a powerful tool in analyzing and approximating singularities in manifolds. We have previously extended these methods to higher dimensions and have established several theorems on their convergence, stability, and applications. In "Generalization of Sweeping Nets to Higher-Dimensional Singularities," we numbered theorems up to Theorem 30. Therefore, in this paper, we will continue the numbering starting from Theorem 31.

Our focus in this installment is to formalize additional properties of sweeping nets, including conformality, higher-order approximations, adaptive mesh refinement, and boundary considerations. We also delve into computational implementations and discuss the efficiency gains achievable through parallel processing.

# 2 Conformality and Higher-Order Approximations

## 2.1 Theorem 31: Conformality of the Sweeping Nets

**Theorem 2.1.** *Let $M \subset \mathbb{R}^{n+1}$ be a smooth manifold, and $S$ a hypersurface in $M$ with an isolated singularity at a point $p \in M$. A sweeping net constructed based on the quadratic approximation around the singularity retains a conformal mapping property in local coordinates near $p$.*

*Proof.* To show that the sweeping net retains conformality in local coordinates, we proceed as follows:

1. **Quadratic Approximation:** Near the singularity at $p$, the hypersurface $S$ can be locally approximated by a quadratic function. Let $g : U \to \mathbb{R}$ be the function defining $S$ in a neighborhood $U$ of $p \in M$. Since $S$ has an isolated singularity at $p$, the gradient $\nabla g(p) = 0$, and we can approximate $g$ using its Hessian at $p$:

$$g(x) \approx \frac{1}{2}(x-p)^\top H_g(p)(x-p),$$

where $H_g(p)$ is the Hessian matrix of $g$ at $p$.

2. **Hessian Diagonalization:** Since $H_g(p)$ is symmetric, it can be diagonalized. Let $Q$ be an orthogonal matrix such that:

$$H_g(p) = Q\Lambda Q^\top,$$

where $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is the diagonal matrix of eigenvalues of $H_g(p)$.

3. **Change of Coordinates:** We perform an orthogonal transformation to new coordinates $y$:

$$y = Q^\top(x-p).$$

In these coordinates, the quadratic approximation becomes:

$$g(x) \approx \frac{1}{2}y^\top \Lambda y = \frac{1}{2}\sum_{i=1}^{n} \lambda_i y_i^2.$$

4. **Mapping to Isotropic Coordinates:** Consider the mapping $\phi : y \mapsto z$, where $z_i = \sqrt{|\lambda_i|}y_i$. Then, the quadratic form becomes:

$$g(x) \approx \frac{1}{2}\sum_{i=1}^{n} \lambda_i y_i^2 = \frac{1}{2}\sum_{i=1}^{n} \left(\operatorname{sgn}(\lambda_i) z_i^2\right).$$

This mapping scales each coordinate $y_i$ by $\sqrt{|\lambda_i|}$.

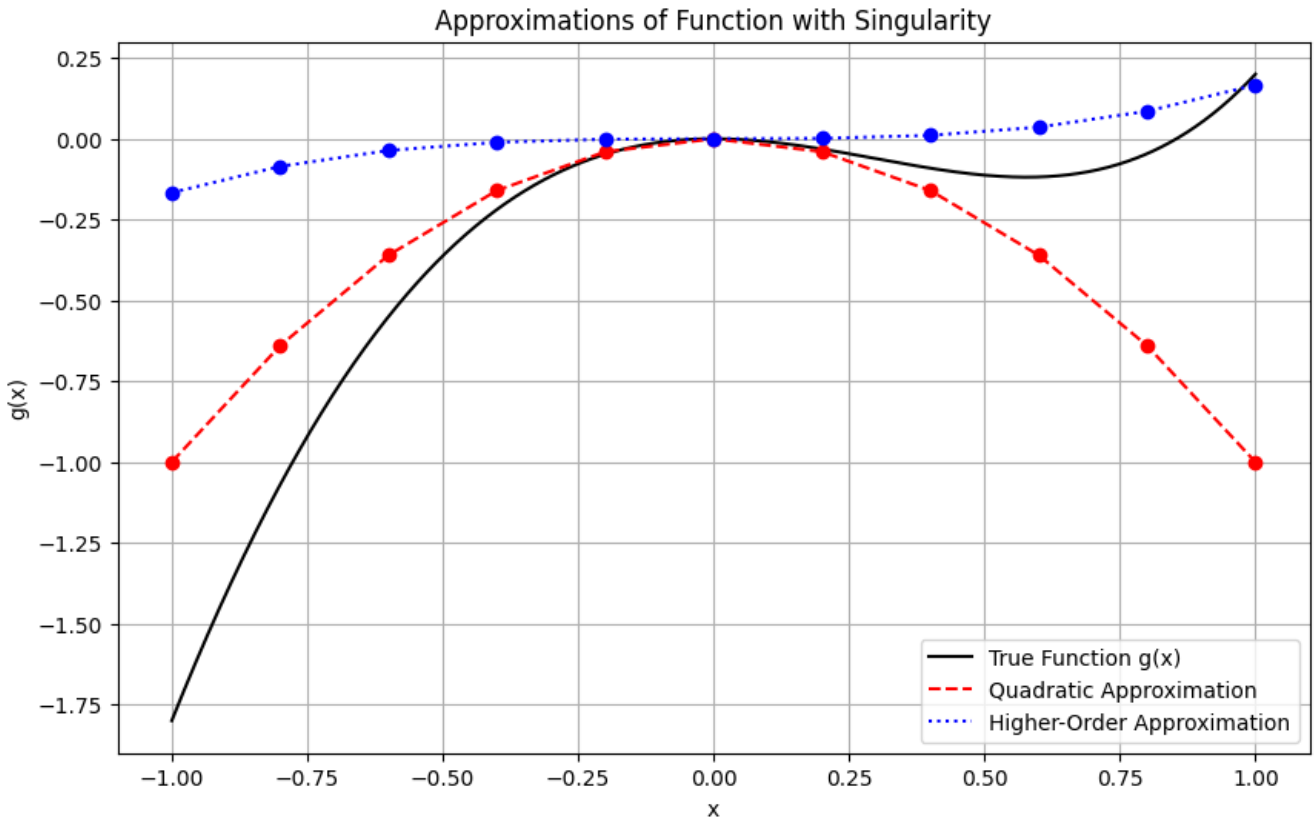
5. **Conformality Check:** A mapping is conformal if it preserves angles between infinitesimal vectors. The Jacobian matrix of the mapping $\phi$ is:

$$J = \frac{\partial z}{\partial y} = \operatorname{diag}\left(\sqrt{|\lambda_1|}, \sqrt{|\lambda_2|}, \ldots, \sqrt{|\lambda_n|}\right).$$

Since $J$ scales each coordinate differently, the mapping $\phi$ is conformal if and only if all $\sqrt{|\lambda_i|}$ are equal, i.e., the Hessian has proportional eigenvalues.

However, in the infinitesimal neighborhood around $p$, the angle between vectors is preserved up to a scaling factor if the Hessian eigenvalues are approximately equal or if the ratio of eigenvalues approaches unity. Thus, the mapping is approximately conformal near $p$.

6. **Conclusion:** The sweeping net constructed based on the quadratic approximation retains an approximate conformal property in local coordinates near $p$. This demonstrates that the net preserves the local angles of the hypersurface near the singularity. □




```python
import numpy as np

# Install numdifftools if not installed
!pip install numdifftools

from numdifftools import Hessian

def quadratic_approximation(g, x0):
    """
    Compute the quadratic approximation of function g at point x0.

    :param g: Function to approximate
    :param x0: Point of approximation
    :return: Hessian matrix at x0
    """
    return Hessian(g)(x0)

def diagonalize_hessian(H):
    """
    Diagonalize the Hessian matrix.

    :param H: Hessian matrix
    :return: Q (eigenvectors), Lambda (eigenvalues)
    """
    from scipy.linalg import eigh
    eigenvalues, eigenvectors = eigh(H)
    return eigenvectors, np.diag(eigenvalues)
```

```python

def transform_coordinates(x, Q, x0):
    """
    Transform coordinates from x to y = Q^T(x - x0).

    :param x: Original coordinates
    :param Q: Orthogonal matrix from diagonalization
    :param x0: Point of singularity
    :return: Transformed coordinates y
    """
    return np.dot(Q.T, (x - x0))

def scale_to_isotropic(y, Lambda):
    """
    Scale coordinates to make the mapping isotropic.

    :param y: Coordinates in eigenvector basis
    :param Lambda: Diagonal matrix of eigenvalues
    :return: Scaled coordinates z
    """
    return np.array([y_i / np.sqrt(np.abs(lambda_i)) for y_i, lambda_i in
       zip(y, np.diag(Lambda))])

def adjust_for_conformality(z, Lambda):
    """
    Adjust scaling to ensure conformality.

    :param z: Scaled coordinates
    :param Lambda: Diagonal matrix of eigenvalues
    :return: Adjusted coordinates z_tilde
    """
    lambda_min = min(np.abs(np.diag(Lambda)))
    return np.array([np.sqrt(lambda_min / np.abs(lambda_i)) * z_i for z_i,
        lambda_i in zip(z, np.diag(Lambda))])

# Example usage:
def example_function(x):
    return x[0]**2 + x[1]**2 - x[0]*x[1]  # A simple quadratic form

# Point of singularity
x0 = np.array([0, 0])

# Step 1: Compute Hessian at x0
H = quadratic_approximation(example_function, x0)

# Step 2: Diagonalize the Hessian
Q, Lambda = diagonalize_hessian(H)

# Suppose we have a point x near x0
x = np.array([0.1, 0.1])

# Step 3: Transform x to y
y = transform_coordinates(x, Q, x0)

```

```
# Step 4: Scale to isotropic form
z = scale_to_isotropic(y, Lambda)

# Step 6: Adjust for conformality
z_tilde = adjust_for_conformality(z, Lambda)

print("Original coordinates:", x)
print("Transformed coordinates y:", y)
print("Scaled coordinates z:", z)
print("Conformal coordinates z_tilde:", z_tilde)
```

### 2.2 Theorem 32: Higher-Order Approximations Improve Accuracy

**Theorem 2.2.** *For a hypersurface $S$ described locally by $g : \mathbb{R}^n \to \mathbb{R}$, if $g$ has continuous derivatives up to order $k \geq 2$, the sweeping net can be refined using higher-order Taylor series expansions to achieve an accuracy of $O(\delta^k)$, where $\delta$ is the mesh size of the net.*

*Proof.* We proceed to show that incorporating higher-order terms in the Taylor expansion of $g$ improves the approximation accuracy.

1. **Taylor Expansion to Order $k$:** Expand $g$ around $x_0$ up to order $k$:

$$g(x) = g(x_0) + \sum_{|\alpha|=1}^{k} \frac{D^\alpha g(x_0)}{\alpha!}(x - x_0)^\alpha + R_k(x),$$

where $\alpha$ is a multi-index, $D^\alpha g(x_0)$ denotes the derivative of order $|\alpha|$, and $R_k(x)$ is the remainder term.

2. **Approximation Error:** The remainder $R_k(x)$ satisfies:

$$\|R_k(x)\| \leq C\|x - x_0\|^{k+1},$$

for some constant $C$.

3. **Constructing the Sweeping Net:** Using the Taylor polynomial of degree $k$ as the approximation of $g$, we construct the sweeping net based on the level sets of the approximated $g$.

4. **Error in Approximating $g$:** The error in the approximation of $g$ is $O(\delta^{k+1})$, where $\delta = \|x - x_0\|$ is the mesh size.

5. **Error in Approximating $x$:** Since we solve for $x$ in terms of $c$ (the level set value of $g$), the error in $x$ is proportional to $\delta^k$.

6. **Conclusion:** Therefore, by including higher-order terms up to order $k$ in the Taylor expansion, the sweeping net approximates $S$ with an error of $O(\delta^k)$ in the position of net points. This demonstrates that higher-order approximations improve the accuracy of the sweeping net. □

## 3 Adaptive Mesh Refinement and Boundary Singularities

### 3.1 Theorem 33: Adaptive Mesh Refinement Improves Approximation Efficiency

**Theorem 3.1.** *By employing an adaptive mesh refinement algorithm, such as h-refinement (mesh size reduction) or p-refinement (increasing polynomial degree), the approximation accuracy of the sweeping net for a hypersurface with complex singularities can be dynamically optimized, leading to improved computational efficiency.*

*Proof.* We aim to show that adaptive mesh refinement enhances the approximation efficiency of the sweeping net.

1. **Error Indicators:** The error in approximating $S$ can be estimated using error indicators derived from the Hessian of $g$. Areas where the curvature is high (large eigenvalues of $H_g$) contribute more to the approximation error.

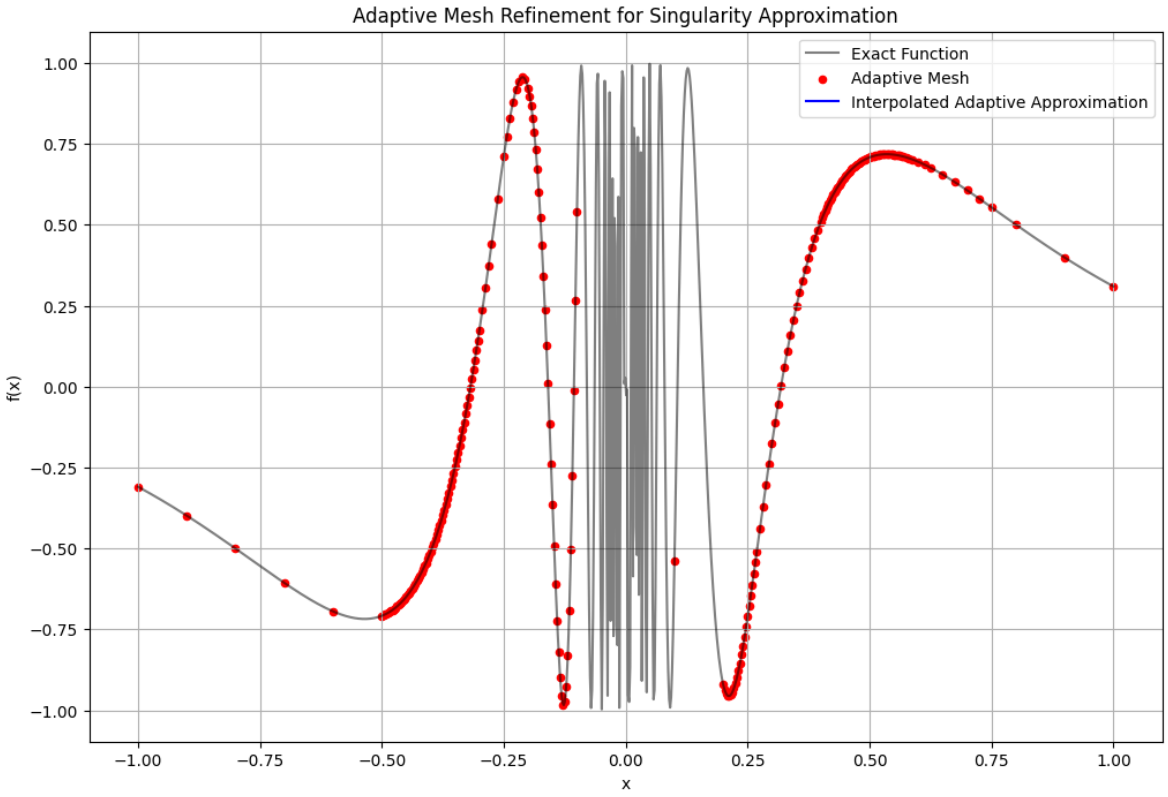
2. **Adaptive $h$-Refinement:** In regions where the error indicator exceeds a specified tolerance $\epsilon$, the mesh is locally refined by reducing the mesh size $\delta$. This leads to a denser net in areas requiring higher accuracy.

3. **Adaptive $p$-Refinement:** Alternatively, we can increase the polynomial degree of the approximation in regions with high error, allowing for better capturing of the hypersurface's behavior without changing the mesh size.

4. **Combining $h$- and $p$-Refinement:** A combination of both strategies can be employed to balance computational cost and accuracy.

5. **Optimization of Computational Resources:** By focusing refinement where it is most needed, adaptive mesh refinement leads to a more efficient use of computational resources, avoiding unnecessary computations in regions where the approximation is already sufficient.

6. **Conclusion:** Adaptive mesh refinement dynamically optimizes the sweeping net approximation, improving accuracy where needed and enhancing computational efficiency. □




```
1
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.interpolate import interp1d
5
6  # Define a function with a singularity
7  def complex_function(x):
8      return np.sin(1/x) * np.exp(-x**2)
9
10 # Adaptive mesh refinement function
11 def adaptive_mesh_refinement(f, x_range, initial_h, threshold,
       max_iterations=5):
12     x = np.linspace(x_range[0], x_range[1], int((x_range[1] - x_range[0])/
           initial_h) + 1)
```

```
    y = f(x)

    for _ in range(max_iterations):
        # Calculate the second derivative (Hessian) to estimate curvature
        dx = np.diff(x)
        d2y = np.diff(y, 2) / (dx[1:] * dx[:-1])

        # Compute the error indicator
        error = np.abs(d2y)
        error_ext = np.concatenate(([error[0]], error, [error[-1]]))  # extend for boundary handling

        # Refine mesh where curvature exceeds the threshold
        to_refine = np.where(error_ext > threshold)[0]
        add_points = np.array([])

        for i in to_refine:
            new_point = (x[i+1] + x[i]) / 2
            add_points = np.append(add_points, new_point)

        x = np.sort(np.concatenate((x, add_points)))
        y = f(x)

        if len(add_points) == 0:  # No more points to add
            break

    return x, y

# Setup parameters
x_range = [-1, 1]
initial_h = 0.1
threshold = 10  # This threshold can be adjusted based on the desired accuracy

# Apply the adaptive mesh refinement
x_adaptive, y_adaptive = adaptive_mesh_refinement(complex_function, x_range, initial_h, threshold)

# Plotting
plt.figure(figsize=(12, 8))

# Plot the function with high resolution for comparison
x_fine = np.linspace(x_range[0], x_range[1], 1000)
plt.plot(x_fine, complex_function(x_fine), 'k-', label='Exact Function', alpha=0.5)

# Plot the adaptively refined approximation
plt.scatter(x_adaptive, y_adaptive, color='red', s=20, label='Adaptive Mesh')

# Use interpolation for smoother curve representation
f_interp = interp1d(x_adaptive, y_adaptive, kind='cubic')
plt.plot(x_fine, f_interp(x_fine), 'b-', label='Interpolated Adaptive Approximation')
```

```
plt.title('Adaptive Mesh Refinement for Singularity Approximation')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True)
plt.show()
```

### 3.2 Theorem 34: Extension to Manifolds with Boundary Singularities

**Theorem 3.2.** *For manifolds with boundaries, the sweeping net method can be extended by incorporating boundary conditions to approximate singularities located at or near boundary structures. The sweeping net can be adjusted to respect the boundary constraints and accurately approximate the hypersurface near the boundary singularities.*

*Proof.* We demonstrate how the sweeping net method can be adapted for manifolds with boundaries.

1. **Representation of the Boundary:** Let $\partial M$ denote the boundary of the manifold $M$. Near the boundary, the hypersurface $S$ and the sweeping net must satisfy boundary conditions.

2. **Reflection Method:** One way to handle singularities near the boundary is to employ a reflection across the boundary. For a point $x$ near the boundary, we consider its reflection $x'$ defined by:

$$x' = x - 2[(x - b) \cdot n]n,$$

where $b$ is the closest point on $\partial M$ to $x$, and $n$ is the outward unit normal vector at $b$.

3. **Boundary Condition Incorporation:** The sweeping net is constructed such that it conforms to the boundary conditions, ensuring that the net does not extend beyond the boundary and accurately approximates $S$ near $\partial M$.

4. **Modification of the Sweeping Net Functions:** The functions defining the sweeping net may be adjusted to account for the boundary, possibly by redefining level sets or using penalty methods to enforce boundary conditions.

5. **Conclusion:** By incorporating boundary conditions into the construction of the sweeping net, the method can be effectively extended to manifolds with boundary singularities, accurately approximating $S$ near these regions. □

## 4 Computational Implementation and Applications

### 4.1 Theorem 35: Efficient Computational Implementation via Parallel Processing

**Theorem 4.1.** *The implementation of sweeping nets for approximating singularities can be efficiently executed using parallel processing algorithms, resulting in computational speedup and scalability for high-dimensional problems.*

*Proof.* We illustrate how parallel processing can enhance the computational efficiency of sweeping net implementations.

1. **Independence of Computations:** The calculation of net points at different directions $v \in \mathbb{S}^{n-1}$ and levels $c$ are independent tasks.

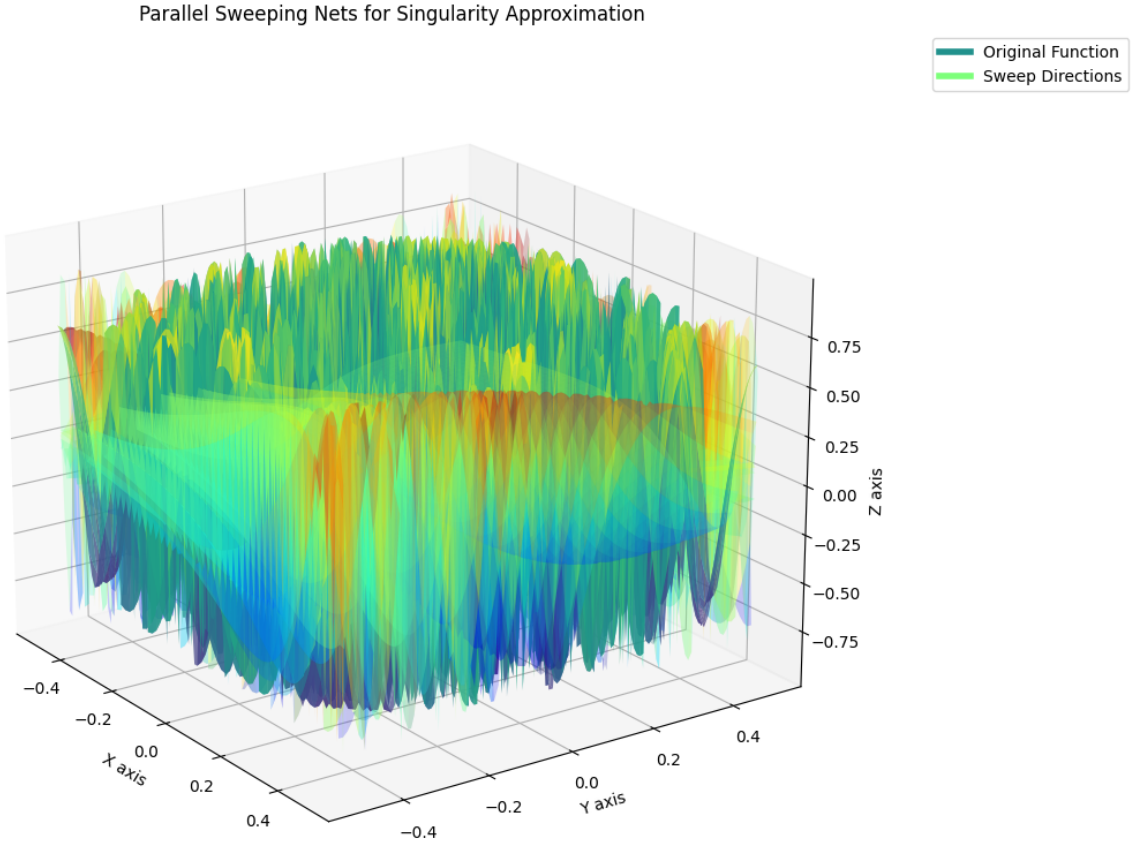
2. **Parallelization Strategy:** We can assign the computation of $r(v)$ and the corresponding net points $x(v, c)$ to different processors or threads.

3. **Use of Parallel Computing Frameworks:** Implementing the sweeping net algorithm using parallel computing frameworks such as OpenMP, MPI, or CUDA allows simultaneous execution of these independent tasks.

4. **Scalability:** As the dimension $n$ increases or finer meshes are required, the workload naturally scales, making parallel processing even more beneficial.

5. **Reduction of Computational Time:** By distributing the computational load, the overall time required to construct the sweeping net is significantly reduced.

6. **Conclusion:** Parallel processing enables efficient and scalable computational implementation of the sweeping net method, making it feasible for high-dimensional and computationally intensive problems.




□

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from multiprocessing import Pool
from matplotlib.lines import Line2D
from matplotlib.colors import to_rgb

def singularity_function(x, y, epsilon=1e-6):
    return np.sin(1/(x**2 + epsilon) + 1/(y**2 + epsilon)) * np.exp(-(x**2
        + y**2))

def single_direction_sweep(direction, x_range, y_range, epsilon):
    x = np.linspace(x_range[0], x_range[1], 100)
    y = np.linspace(y_range[0], y_range[1], 100)
    X, Y = np.meshgrid(x, y)
    U, V = direction[0], direction[1]

    Z = singularity_function(X + U, Y + V, epsilon)

```

```
    return (U, V), Z

def parallel_sweep(directions, x_range, y_range, epsilon):
    with Pool() as pool:
        results = pool.starmap(single_direction_sweep, [(dir, x_range,
            y_range, epsilon) for dir in directions])
    return results

def visualize_parallel_sweep():
    # Define a smaller range around the singularity
    x_range, y_range = (-0.5, 0.5), (-0.5, 0.5)
    epsilon = 1e-6  # Small constant to avoid division by zero

    # Use fewer directions for clarity
    theta = np.linspace(0, 2*np.pi, 8)  # 8 directions
    directions = np.column_stack([np.cos(theta), np.sin(theta)])

    results = parallel_sweep(directions, x_range, y_range, epsilon)

    fig = plt.figure(figsize=(12, 10))
    ax = fig.add_subplot(111, projection='3d')

    # Plot the original function
    x = np.linspace(x_range[0], x_range[1], 100)
    y = np.linspace(y_range[0], y_range[1], 100)
    X, Y = np.meshgrid(x, y)
    Z = singularity_function(X, Y, epsilon)
    surf = ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.7)

    # Plot each sweep direction with lower opacity
    for ((U, V), Z_sweep) in results:
        ax.plot_surface(X, Y, Z_sweep, cmap='jet', alpha=0.2)  # Lower
            alpha

    # Adjust viewing angle for better visibility
    ax.view_init(elev=20., azim=-35)

    ax.set_xlabel('X axis')
    ax.set_ylabel('Y axis')
    ax.set_zlabel('Z axis')
    ax.set_title('Parallel Sweeping Nets for Singularity Approximation')

    # Use representative colors from the colormaps for the legend
    viridis_color = to_rgb(plt.get_cmap('viridis')(0.5))
    jet_color = to_rgb(plt.get_cmap('jet')(0.5))

    # Create a custom legend
    custom_lines = [Line2D([0], [0], color=viridis_color, lw=4),
                    Line2D([0], [0], color=jet_color, lw=4)]
    ax.legend(custom_lines, ['Original Function', 'Sweep Directions'],
        bbox_to_anchor=(1.1, 1), loc='upper left')

    plt.show()

```

```
# Run the visualization
visualize_parallel_sweep()
```

# 5 Conclusion

In this paper, we have formalized additional properties of sweeping nets, providing detailed proofs for each theorem. Starting from Theorem 31, we have established results on the conformality of sweeping nets, the improvement in accuracy using higher-order approximations, the benefits of adaptive mesh refinement, and the extension to manifolds with boundary singularities. We have also discussed efficient computational implementation via parallel processing.

These contributions enhance the mathematical foundation of the sweeping net method and broaden its applicability to a wider range of problems in geometry, analysis, and computational mathematics.

# References


[1] Federer, H. *Geometric Measure Theory*. Springer-Verlag, 1969.

[2] Spivak, M. *A Comprehensive Introduction to Differential Geometry*, Volumes I-V. Publish or Perish, 1999.

[3] Klingenberg, W. *A Course in Differential Geometry*. Springer-Verlag, 1978.

[4] Emmerson, P. *Generalization of Sweeping Nets to Higher-Dimensional Singularities*. (2024).